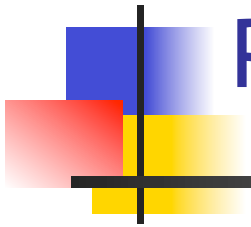


Refinement-Based Context-Sensitive Points-To Analysis for Java

Manu Sridharan, Rastislav Bodík
UC Berkeley

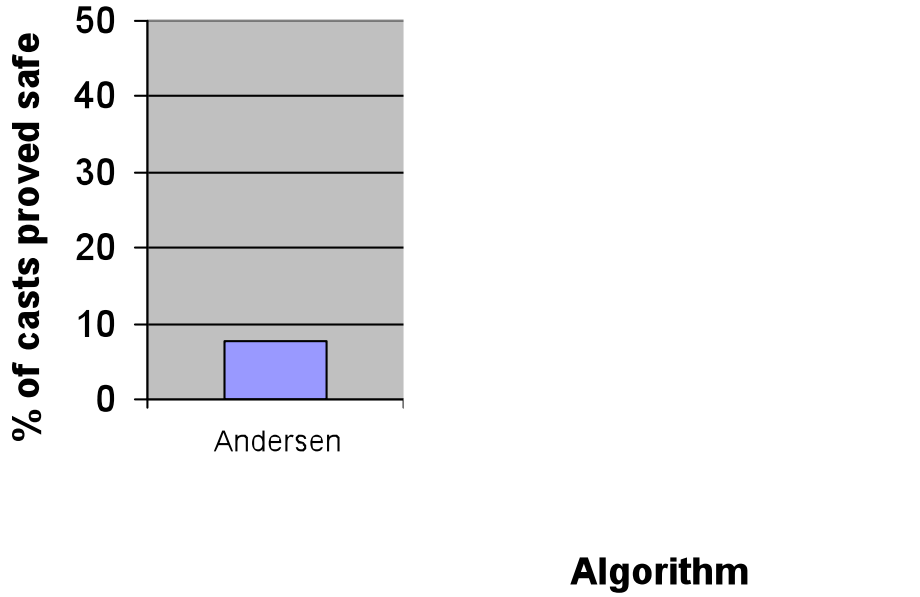
PLDI 2006

What Does Refinement Buy You?

Increased scalability: enable new clients

- Memory: orders of magnitude savings
- Time: answer for a variable comes back in 1 second
-) Suitable for IDE

Precision:



Approach: Focus on the Client

Demand-driven: only do requested work

Client-driven refinement: stop when client satisfied

Example:

- client asks: “can x point to o?”
- we refine until we answer NO (the good answer) or we time out

Context-Sensitive Analysis Costly

Context-sensitive analysis (def):

- Compute result as if all calls inlined
- But, collapse recursive methods

Exponential blowup (code growth)

Why Not Existing Technique?

Most analyses approximate same way in all code

- E.g., k-CFA
- Precision lost, esp. for data structures

Our analysis focuses precision where it matters

- Fully precise in the limit
- Only small amount of code analyzed precisely
- First refinement algorithm for Java

Points-To Analysis Overview

Compute objects each variable can point to

For each var x, points-to set pt(x)

Model objects with abstract locations

1: x = new Foo() yields pt(x) = { o_1 }

Flow-insensitive: statements in any order

Points-To Analysis as CFL-Reachability

1) Assignments

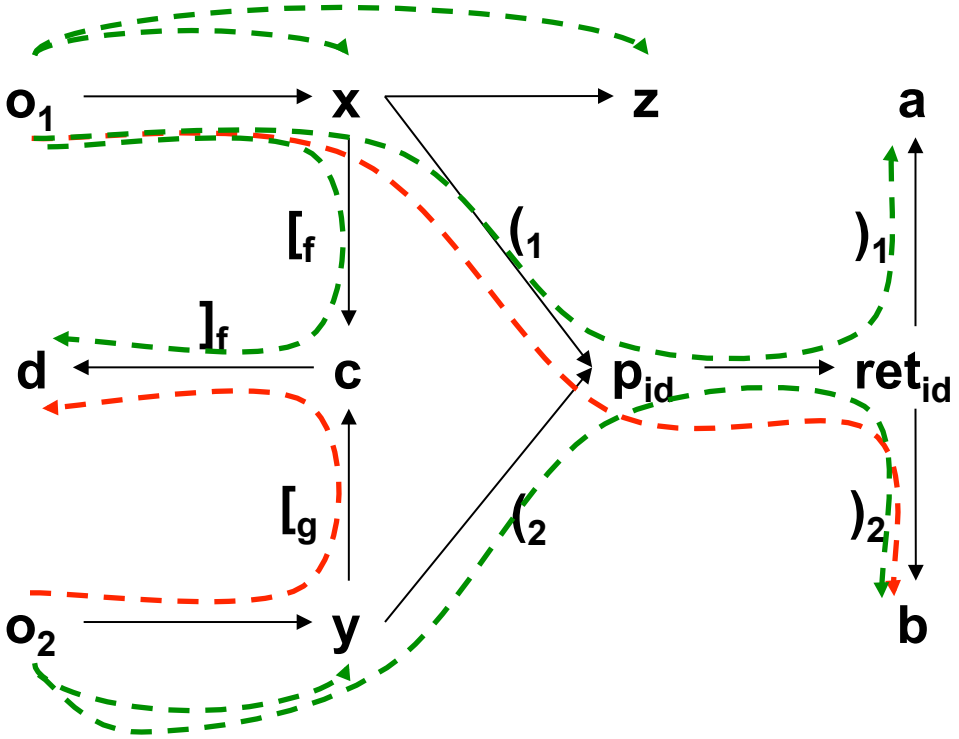
x = new Obj(); // o_1
y = new Obj(); // o_2
z = x;

2) Method calls

id(p) { return p; }
a = id(x);
b = id(y);

3) Heap accesses

c.f = x;
c.g = y;
d = c.f;



pt(x) = { o | o *flowsTo* x }

flowsTo: balanced parentheses

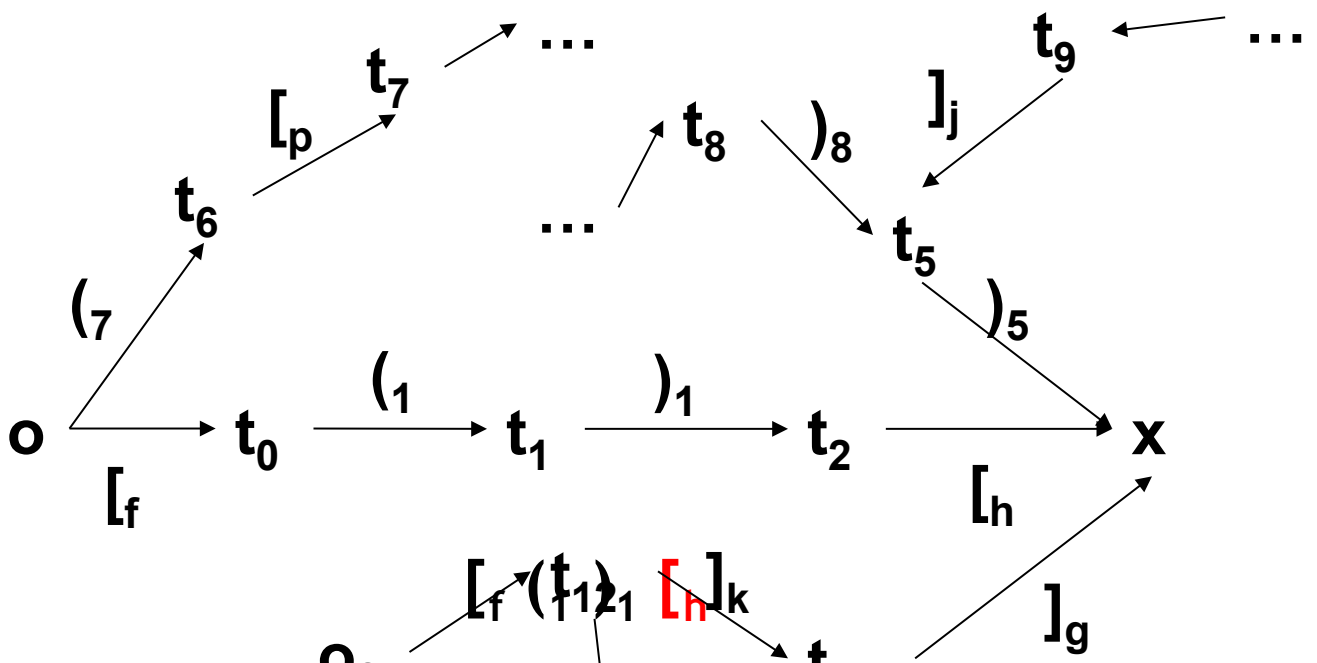
Summary of Formulation

Graph represents program

Compute reachability with two filters

- Language of balanced call parens
- Language of balanced field parens

Single path problem

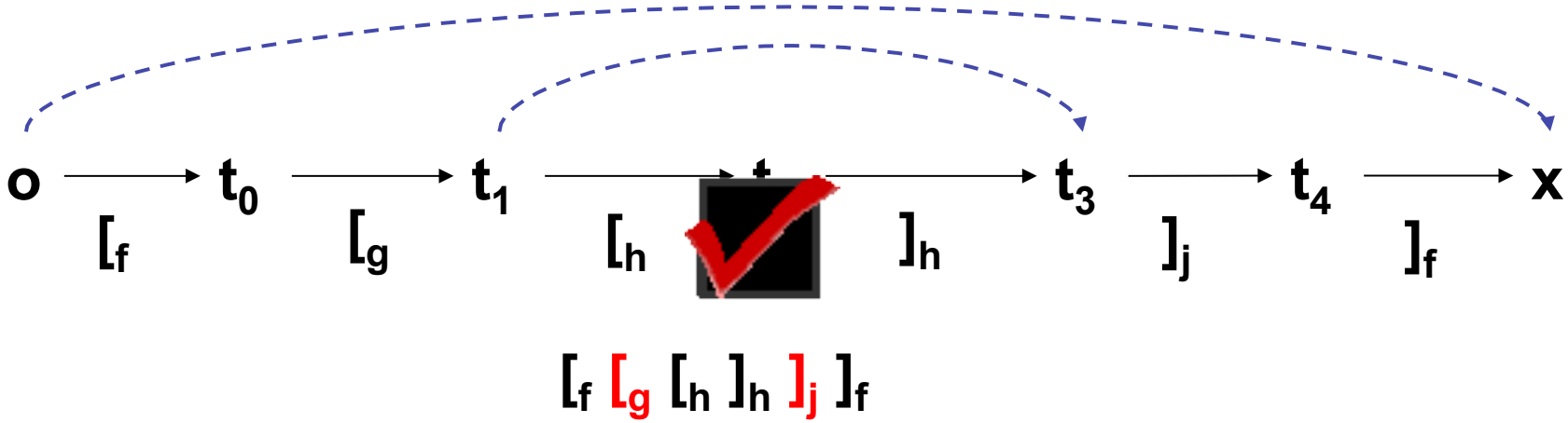


Problem: show path is unbalanced

Goal: reduce number of visited edges

Insight: enough to find one unbalanced paren

Approximation via Match Edges

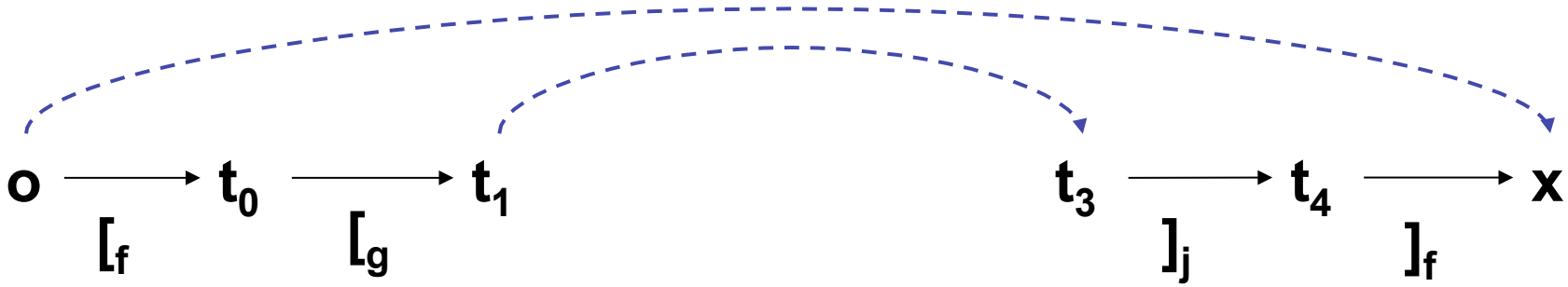


Match edges connect matched field parens

- From source of open to sink of close
- Initially, all pairs connected

Use match edges to skip subpaths

Refining the Approximation



$[_f\ [_g\ [_h\ ]_h\ ]_j\ ]_f$

Refine by removing some match edges

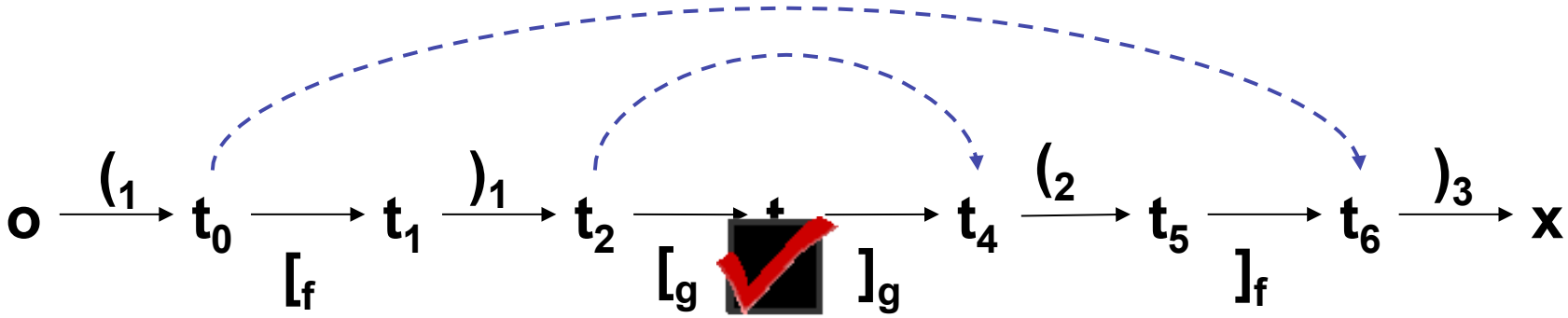
- Exposes more of original path for checking

Soundness: Traverse match edge)
assume field parens balanced on skipped path

Remove where unbalanced parens expected

- Explore deeper levels of pointer indirection

Refinement With Both Languages



Fields: $[_f\ [_g\ ]_g\ ]_f$

Calls: $(_1\ )_1\ (_2\ )_3$

Match edges enable approximation of calls

- Only can check calls on match-free subpaths

Match edge removal) more call checking

- Key point: refine heap and calls together

Evaluation

Experimental Configuration

Implemented in Soot framework

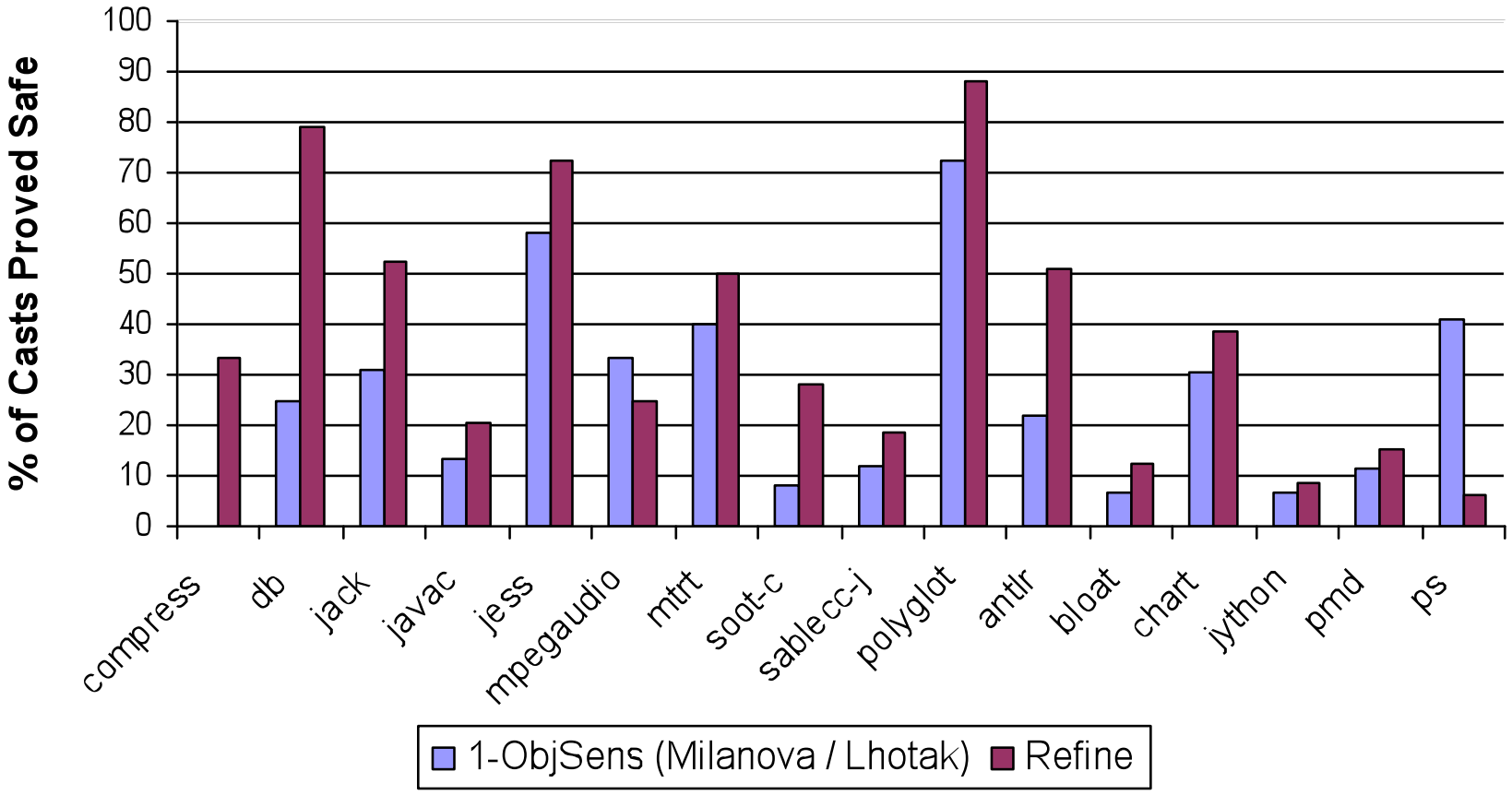
Tested on large benchmarks **x** 2 clients

- SPECjvm98, Dacapo suite
- Downcast checking, factory method props

Refine context-insensitive result

Timeout for long-running queries

Precision: Cast Checking



Scalability: Time and Memory

Average query time less than 1 second

- Interactive performance (for IDE)
- At most 13 minutes for casts,
  4 minutes for factory client

Very low memory usage: at most 35MB

- Of this, 30MB for context-insensitive result
- Compare with >2GB for 1-ObjSens analysis

Demand-Driven vs. Exhaustive

Demand advantage: no caching required

- Hence, low memory overhead
- No engineering of efficient sets
- Good for changing code; just re-compute

Demand advantage: faster for many clients

- Often only care about some variables

Demand disadvantage: slower querying all vars

- At most 90 minutes for all app. vars
- But, still good precision, memory

Conclusions

Novel refinement-based analysis

- More precise for tested clients
- Interactive performance for queries
- Low memory: could scale even more
- Relatively easy to implement

Insight: refine heap and calls together

- Useful for other balanced-paren analyses?